Patent Application
Atty. Docket No. 00100.03.0040

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

FILING OF A UNITED STATES PATENT APPLICATION

METHOD AND APPARATUS FOR STATIC SINGLE ASSIGNMENT FORM DEAD CODE ELIMINATION

Inventors:

Norman Rubin 22 Essex Street Cambridge, Massachusetts 02139 USA	Myron King 55 Plymouth Street, #3 Cambridge, Massachusetts 02141 USA

Assignee:

ATI Technologies, Inc.
1 Commerce Valley Drive East
Markham, Ontario
Canada L3T 7X6

Attorney of Record:
Timothy J. Bechen
Registration No. 48,126
Vedder, Price, Kaufman & Kammholz, P.C.
222 North LaSalle Street, Suite 2600
Chicago, Illinois 60601
Phone (312) 609-7500
Fax (312) 609-5005

Express Mail Label No. EV 320520585 US

Date of Deposit: January 28, 2004
I hereby certify that this paper is being deposited with the U.S. Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. Section 1.10 on the date of deposit, indicated above, and is addressed to: Commissioner for Patents, Mail Stop Patent Application, P.O. Box 1450, Washington, DC 20231.

Name of Depositor: Karenina Oliver

Signature: Karenina Oliver

METHOD AND APPARATUS FOR STATIC SINGLE ASSIGNMENT FORM DEAD CODE ELIMINATION

FIELD OF THE INVENTION

[0001] The present invention relates generally to computer program compilers and more specifically to optimization of machine language code through the utilization of dead code elimination of instructions in static single assignment form.

BACKGROUND OF THE INVENTION

[0002] In a computer system, a compiler is utilized to convert a software program in a programming language into machine language. A processor then may execute the machine language to perform the operations designated by the software program. Although, inefficiencies arise when using compilers due to the presence of executable instructions within the machine language whose results do not affect program outputs.

[0003] One solution to overcome inefficiencies is the elimination of dead code, wherein dead code is executable program instructions whose results do not affect program output. A common scheme for removing dead code is based on program representations called static single assignment form (SSA). The SSA form can be briefly described as a form where each definition of a variable is given a unique version, and different versions of the same variable can be regarded as different program variables. Each use of a variable version can only refer to a single universal definition. When several definitions of a variable reach a common node, typically referred to as a merging node, in the control flow graph of the program, a phi function assignment statement, $a_n=\Phi(a_1, a_2,\ldots a_m)$, is inserted to merge the variables into the definition of a new variable version a_n. Thus, the semantics of single reaching definitions are maintained.

[0004] The SSA based dead code elimination may be performed by adding a single bit to each instruction, wherein that bit is designated as a live bit and initially all live bits are set to

false. The next step is to identify a set of critical instructions, wherein critical instructions are instructions that directly produce a program output. As long as a worklist of instructions is not empty, select an instruction from the worklist. Thereupon, walking backwards over SSA links, from use to definition, the next step is to identify all instructions that are needed as a source of the originally selected instruction. If the instruction needed for the originally selected instruction is not live, the instruction then is marked as live and added to the worklist.

[0005] The process is repeated for all instructions on the worklist. Once the worklist is empty, the value of the live bit determines if the instruction is live or dead. This provides an efficient scheme, since adding an instruction to the worklist can occur at most only one time and also designates all dead code. Although, this current approach is limited because it can only provide for the deletion of scalar instructions and fails to detect dead components. This approach is described in further detail in R. Cytron et al., *Efficiently computing static single assignment form an the control depend graph*, ACM Trans. On Programming Languages and Systems, 13(4), pp. 451-490, October 1991.

[0006] Another approach for the elimination of dead code is based on extending the single live bit to a set of bits based on the computation being performed with a specific number of bits as disclosed in U.S. Patent No. 6,571,387 entitled "Method and computer program product for global minimization of sign-extension and zero-extension operations. The number of bits may not be supported be a host processor, such as a host processor might require computations be done in sixty-four (64) bit integers, while the program instructions have been specific to thirty-two (32) bits integers. In this approach, a compiler may insert instructions to compensate for this difference. If a compiler can determine that all of the upper bits are dead code, the extra inserted instructions may be deleted.

[0007] This technique extends SSA based dead code elimination by adding a bit field to each instruction of the maximum size of a generated value. Using this bit mask, the nth bit that is used in the instruction will be set to a value of 1. During the backward walk over the SSA links, the process updates the bits in each source instruction. Therefore, once all of the SSA links have been examined, each instruction has been marked with the exact bits used. If an instruction is a sign or zero extension instruction, and the upper bits are not marked, then the instruction may be dead and thereby eliminated. This technique is limited to dead code operations that consist of sign or zero extension instructions.

[0008] In a single instruction multiple data (SIMD) processing environment, there advantages to using a superword register, wherein a superword register includes a hardware resource that can hold a small, but more than one, number of words of data. In one exemplary orientation, the superword register can hold up to 128 bits divided into four floating point components.

[0009] Instructions that operate on superword registers operate in parallel on all components and therefore are capable of achieving very high performance provided that more than one component contains data. If a program computes an unused value in a superword component, then the value of the component is said to be dead, whereas the other elements are termed live. Some compilers using superword registers fail to eliminate the dead code and therefore reduce operating efficiency due to extra un-needed components. For example, the disclosure of U.S. Patent No. 6,571,387 does not provide for dead code elimination using a superword register. Superword registers provide improved processing times in a SIMD environment and there does not exist a dead code elimination technique with a superword register. Therefore, there exists a need for removing dead code from a superword register.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] FIG. 1 illustrates a flowchart of a method for dead code elimination in accordance with one embodiment of the present invention;

[0011] FIG. 2 illustrates a flowchart of a method for dead code elimination in accordance with another embodiment of the present invention;

[0012] FIG. 3 illustrates a schematic block diagram of an apparatus for dead code elimination in accordance with one embodiment of the present invention;

[0013] FIG. 4 illustrates a block diagram an instruction for use with a superword register in accordance with one embodiment of the present invention; and

[0014] FIG. 5 illustrates a graphical representation of a plurality of decision trees of instructions and the elimination of dead code in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

[0015] Briefly, a method and apparatus for dead code elimination includes adding to each instruction (1) a set of bits called a write mask (one per component) used to indicate the components produced by the instruction, (2) a link called the "previous write" linking the instruction with the prior instruction writing components of the same superword, and (3) a set of bits (one per component) called live bits, indicating the components consumed by subsequent uses of the superword register and the instruction is in SSA form. Initially all live bits are set to false.

[0016] Instructions are processed from a work list. The instruction may be any suitable executable instruction, such as but not limited to a mathematical instruction, for example

addition, subtraction, multiplication, division, an equivalence instruction, for example a equals b, or any other suitable instruction found within a programming language for conversion by a compiler into a machine level language. Furthermore, a worklist is any suitable collection of instructions. The next step is to identify a set of critical instructions, wherein critical instructions are instructions that directly produce a program output. All critical instructions are placed on the work list.

[0017] As each instruction I is removed from the worklist, a backward walk over the ssa links and partial write links, can be used to identify each instruction J which produce a source used by I. Based on the live bits in I, the corresponding live bits of J may be increased. If I uses a component produced by J and the live bits of J for that component are not set, then set the live bit for that component in instruction to on, and add J to the worklist.

[0018] The preceding computation is repeated until the worklist is empty.

[0019] In the method and apparatus, each of the instructions include a previous link, a list of live components and a write mask, where a previous link connects instructions that write different components of the same resource and the write mask is an n-bit field where n is determined by the number of elements of a superword register. For example, if a superword register allows for four components, this provides for a write mask having four bits, such as an instruction associated with an x, y, z and w field.

[0020] The method and apparatus further includes determining if all elements within a particular resource are required for the at least one second instruction. As discussed above, in one embodiment having 4 elements, a determination is made if the x field is required, if the y field is required, if the z field is required and if the w field is required. If all the elements are required, the method and apparatus provides for deleting the first instruction as it is determined

that this instruction is extraneous, otherwise referred to as dead code. Therefore, the method and apparatus provides for elimination of dead code by examining the steps of the instructions in accordance with the parameters of the SSA based instructions.

[0021] FIG. 1 illustrates the steps of one embodiment of a method for eliminating dead code. The method begins, step 100, by examining a first instruction off of a worklist, wherein the first instruction includes a previous link, live bits and a write mask, step 102. A standard instruction includes a first term and a second term, wherein the instructions disposed on the worklist have the three additional fields added thereto. The previous link is a field that indicates a previous instruction and the write mask allows for operation and coordination of instructions with a superword register. The live bits indicate components produced by this instruction or by a prior instruction that can be reached via the previous write link. As the superword register has a plurality of elements, the write mask coordinates instructions with specific components. This allows for operation of compiler operations such as a swizzle and maintenance of element-specific based calculations.

[0022] The first instruction is examined to determine if any of the elements are live for a particular calculation. If a particular element is not live, this indicates that this instruction is dead code and should be eliminated from the generated machine code. Step 104 is examining one or more second instructions, wherein the one or more second instructions are source instructions of the first instruction and each of the at least one second instructions include a previous link, a set of live bits and a write mask. The previous link, live bits and the write mask of the one or more second instructions have the same structure of the previous link, live bits and the write mask as the first instruction pulled off of the worklist.

[0023] Step 106 is determining if all elements within a particular field are required for the at least one second instruction. This step is performed by looking at each element. In the embodiment having four elements, the x element is examined, the y element is examined, the z element is examined and the w element is examined to determine if the element is live. Also noting in the method and apparatus that with the operation of a swizzle function, the various elements may be representative of other specified elements.

[0024] Step 108 is deleting the first instruction from the machine code if no components are live. If no components are live, this means that this particular instruction is extraneous because result of this instruction are not used by another instruction. Thereupon, the method is complete, step 110.

[0025] FIG. 2 illustrates a further embodiment of a method for eliminating dead code. Similar to FIG. 1, this method is performed by a compiler operating to convert a program language software code into a machine language code, otherwise referred to as compiling the program language software. The method begins 120, by receiving a plurality of instructions, step 122. These instructions are in SSA form.

[0026] Step 124 is adding to each instruction a previous link. The previous link indicates a previous instruction that produces for various components within a multiple component resource. Step 126 is adding to each instruction a write mask. As discussed above, the write mask is an n-bit field and n is the number of components in a superword register. Step 128 is generating a worklist by writing only critical instructions to the worklist. This step is accomplished by examining each instruction and determining if the instruction directly produces a program output, then it is deemed critical.

[0027] Step 130 is setting a live bit for each component of the plurality of instructions. In one embodiment, this live bit is an n-bit field, where n is the number of components in the superword register and the live bit is true if it is known that the specific component, for example x, y, z or w, is not dead. Thereupon, step 132 of the method is examining a first instruction off of the worklist. In one embodiment, the step of examining the instruction includes walking backwards over the SSA links, steps defining particular instructions relative to a product, as discussed in greater detail in FIG. 5. The examination includes identifying all instructions that are needed as the source of the first instruction. Based on the operation of the instruction, any input swizzles and the write mask of the instruction, the examination includes identifying the bits that are required from a particular instruction.

[0028] Step 134 is examining further instructions off of the worklist. This method is an iterative process looking at multiple instructions to determine which instructions are extraneous, dead code. Therefore, step 136 is determining if all elements within a particular field are required for the subsequent instructions, the at least one second instructions. If no elements are required, step 138 is deleting the first instruction from the machine program, as this instruction is dead code. Thereupon, the method is complete, step 140.

[0029] It should also be noted that in the present invention, the maximum number of times any single instruction may be added to the worklist is the number of components within the superword register based on the rate of convergence. For example, if the superword register has four components, x, y, z and w, the instruction may be added only once for each component, and not necessarily the corresponding component but rather the component allocation due to swizzling. In other words, if a swizzle operation allocated the x component to the y component, the instruction may be added to the worklist based on the y component that is actually the x

component, wherein the instruction may have been previously added to the list based on the x component.

[0030] FIG. 3 illustrates an apparatus for the elimination of dead code. A processor 150 is operatively coupled to a memory 152. The memory stores executable instructions 154 therein. The processor 150 may be, but not limited to, a single processor, a plurality of processors, a DSP, a microprocessor, an ASIC, a state machine, or any implementation capable of processing and executing software. The term processor should not be construed to refer exclusively to hardware capable of executing software, and may implicitly include DSP hardware, ROM for storing software, RAM, and any other volatile or non-volatile storage medium. The memory with executable instructions 152 may be, but not limited to, a single memory, a plurality of memory locations, shared memory, CD, DVD, ROM, RAM, EEPROM, optical storage, microcode or any other non-volatile storage capable of storing digital data for use by the processor 150.

[0031] The executable instructions 154 are provided to the processor 150 such that the processor 150 performs operations in response thereto. In one embodiment, the processor 150 performs compiler operations to convert programming language instructions into machine language instructions. The processor 150 is further operative to perform the steps of the methods illustrated in FIGS. 1 and 2 and processes discussed above as associated with the methods for dead code elimination. Therefore, the methods discussed above may be fully implemented and executed by the processor 150 in response to the executable instructions 154.

[0032] In one embodiment of the present invention, the processor 150 is further coupled to one or more superword registers 156, wherein each superword register may be a single hardware resource capable of holding a limited number of words of data. The processor 150,

through performing the method steps discussed above with regards to FIGS. 1 and 2, reads and writes register values 158 to the superword register for tracking the status of the various superword components.

[0033] Thereupon, once the processor 150 compiles program code software into machine language, the compiled instructions are then designated on a component level, for example x, y, z, and w. In the SIMD environment, the instructions may then be efficiently provided to their corresponding processors, such as an x component processor, a y component processor, a z component processor and a w component processor, using the superword register.

[0034] FIG. 4 illustrates a graphical representation of an instruction 170, in accordance with one embodiment of the present invention. The instruction includes the two elements of the instruction, S1 172 and S2 174. Although, in the present invention the previous link field 176 is added, as well as the write mask 178. The added write mask 178 allows for usage of the instruction with the superword register based on tracking the superword register components. It should also be noted that in another embodiment, the instruction further includes a live bit, as discussed above.

[0035] FIG. 5 illustrates a graphical representation of a plurality of instructions that a typical compiler would optimize using the method and apparatus for eliminating dead code of the present invention. The illustrated example include 3 exports, 200, 202 and 204, wherein the exports are some output value, such as a computed value computed by the natural progression of the steps from top to bottom.

[0036] For example, walking backwards across SSA links, export 1 200 is the result of instruction 1A 210. 1A 210 is the result of the combination of 2A 212 and 2B 220. Following the SSA links, 2A is the result of an operation with just 3A 214, which is the result of 4A 216

and 4B 218. Back down SSA links, 2B 220 is the result of 3B 222, and this fully covers this illustrative example of the steps subsequent to the production of export 1 200.

[0037] In the elimination of dead code and the instructions illustrated in FIG. 5 having SSA form, the worklist is utilized to identify steps that are dead code. Therefore, in examining export 2 202, this is the result of 1B 240. 1B 240 is the result of an operation with 2C 242 and 2D 248. Walking up the SSA links, element 2C 242 is the result of 3B 222 and 3C 244. As 3B does not have any higher instructions, step 3C is examined as the result of 4C 246. Back down the SSA links, the second element for 1B 240 was 2D 248, which is the result of 3D 250, wherein 3D is the result of 4C 246 and 4D 252.

[0038] Following the progression of the third export 204, 1C 260 is based on 2D 248 and 2E 262. The instructions for 2D 248 were previously determined with respect to the second export 202, therefore, the steps above 2D may be removed as dead code using the previous bit, the write mask and in one embodiment the live bit for the SSA form instructions. With respect to 2E 262, this is the result of 3E 264 which is the result of 4D 252 and 4E 266. In the event there are any further instructions above 4D 252, these would have been determined upon inspection of the second export 202, therefore these instructions could also be eliminated as dead code.

[0039] As such, the present invention provides for the elimination of dead code in an SSA form instruction. Through the inclusion of the previous link value and the write mask, the instructions may be processed with a superword register for operation a SIMD environment. Therefore, through the modified instructions, the superword register may be utilized and data may be more readily provided to SIMD processor executing machine language operating instructions.

[0040] It should be understood that the implementation of other variations and modifications of the invention in its various aspects will be apparent to those of ordinary skill in the art and that the invention is not limited by the specific embodiments described herein. For example, the superword register may contain any suitable number of components and thereby the write mask contains the corresponding number of bits to allow for the effect management of any number of components in compiler operations, such as swizzles. It is therefore contemplated to cover by the present invention, any and all modifications, variations, or equivalents that fall within the spirit and scope of the basic underlying principles disclosed and claimed herein.